Machine Learning Project Report

Alessandro Romeo

May 26, 2024

Abstract

This report presents the implementation and analysis of machine learning models, including data preprocessing techniques like PCA, LDA, and Z-normalization. The project aims to evaluate the performance of these models using different metrics such as DCF and minimum DCF. The results are compared and discussed to determine the best model and preprocessing method.

Contents

Introduction

This report covers the analysis and implementation of logistic regression models and various preprocessing techniques on a dataset. The objective is to assess the impact of these techniques on model performance and to determine the best-performing model for different scenarios.

1 Data Preprocessing

1.1 Description

The project task consists of a binary classification problem. The goal is to perform fingerprint spoofing detection, i.e. to identify genuine vs counterfeit fingerprint images. The dataset consists of labeled samples corresponding to the genuine (True, label 1) class and the fake (False, label 0) class. The samples are computed by a feature extractor that summarizes high-level characteristics of a fingerprint image. The data is 6-dimensional.

The training files for the project are stored in file `Project/trainData.txt`. The format of the file is the same as for the Iris dataset, i.e. a csv file where each row represents a sample. The first 6 values of each row are the features, whereas the last value of each row represents the class (1 or 0). The samples are not ordered.

Load the dataset and plot the histogram and pair-wise scatter plots of the different features. Analyze the plots.

1. Analyze the first two features. What do you observe? Do the classes overlap? If so, where? Do the classes show similar mean for the first two features? Are the variances similar for the two classes? How many modes are evident from the histograms (i.e., how many "peaks" can be observed)?

2. Analyze the third and fourth features. What do you observe? Do the classes overlap? If so, where? Do the classes show similar mean for these two features? Are the variances similar for the two classes? How many modes are evident from the histograms?

3. Analyze the last two features. What do you observe? Do the classes overlap? If so, where? How many modes are evident from the histograms? How many clusters can you notice from the scatter plots for each class?

1.2 Code

```
# Plot histogram and scatter plots
plot_hist(D, L, 'hist_feature')
plot_scatter(D, L, 'scatter_features')

for cls in [False, True]:
```

```
    Dcls = D[:, L==cls]
    mucls = mcol(Dcls.mean(1))
    Ncls = float(Dcls.shape[1])
    DCcls = Dcls - mucls
    Ccls = 1/Ncls*DCcls@DCcls.T
    varcls = Dcls.var(1)
    stdcls = Dcls.std(1)
```

1.3 Results

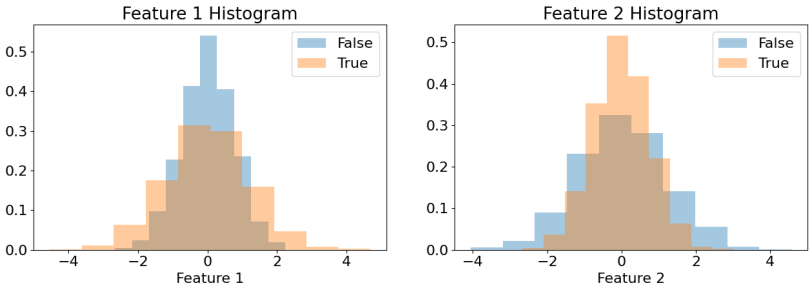

Figure 1: Histograms of Features 1 and 2

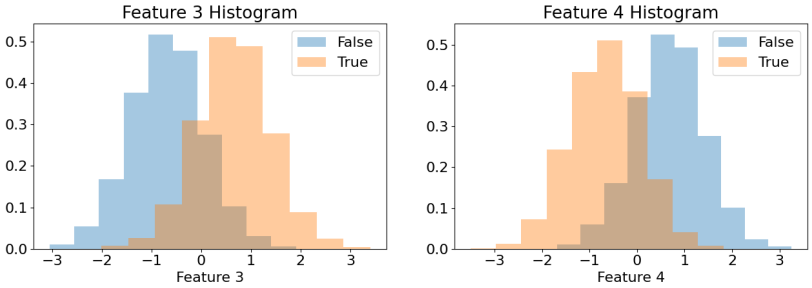

Figure 2: Histograms of Features 3 and 4

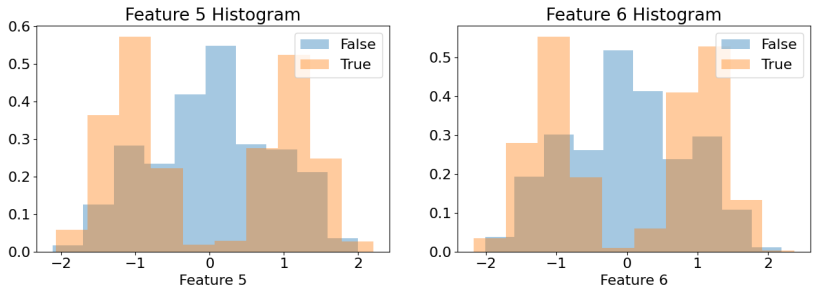

Figure 3: Histograms of Features 5 and 6

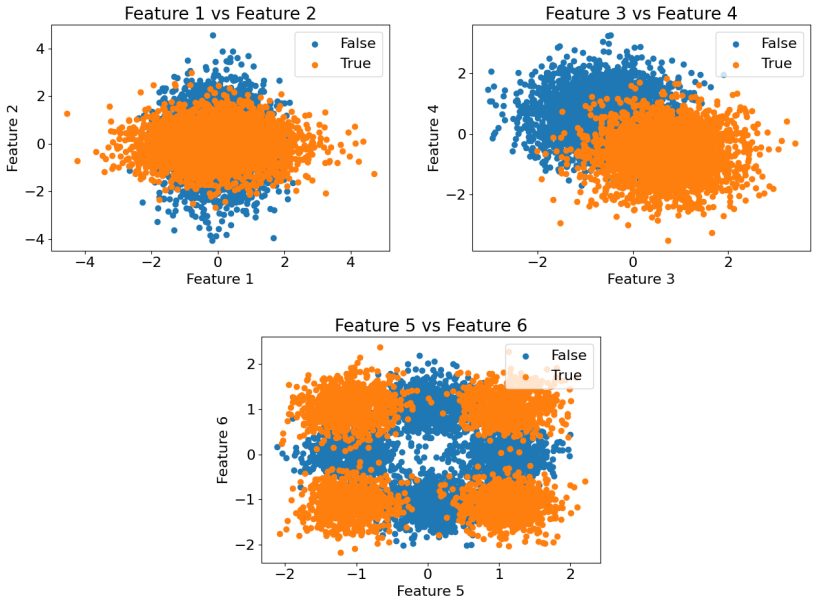

Figure 4: Scatter Plots of Features 1 and 2, Features 3 and 4, Feature 5 and 6

1.4 Observations

- **Scatter Plot Analysis**:
  - The scatter plot indicates areas of overlap between the classes, suggesting regions where the feature values for both classes are similar.
  - This overlap signifies that the features are not perfectly separable for the given classes, making classification more challenging in these regions.
- **Histogram Plot Analysis**:

  - The histogram plot shows the number of peaks for each feature, indicating the distinct modes in the data distribution.
  - Multiple peaks suggest the presence of subgroups or clusters within the same class, indicating a more complex data structure.

- **Observations on Class Statistics**:
  - Means: The means for each feature in both classes are different, indicating that the central tendency of the features differs between classes.
  - Variances: The variance for each feature also varies between classes. For example, the first feature has a higher variance in the True class compared to the False class.
  - Standard Deviations: Similar to variance, the standard deviation indicates how spread out the features are. The True class has a higher spread for some features compared to the False class.
  - Covariances: The covariance matrices indicate the relationship between features within each class. The covariances are different for each class, showing that the relationship between features differs based on the class.

- **Conclusion**:
  - The classes have different mean and variance values for the features, which helps in distinguishing them.
  - However, the presence of overlaps in the scatter plots and the peaks in the histogram plots indicate that there are areas where the classes are not perfectly separable.
  - This analysis helps in understanding the data distribution and can guide further model selection and feature engineering steps.

2 PCA and LDA

2.1 Description

Apply PCA and LDA to the project data. Start analyzing the effects of PCA on the features. Plot the histogram of the projected features for the 6 PCA directions, starting from the principal (largest variance). What do you observe? What are the effects on the class distributions? Can you spot the different clusters inside each class?

Apply LDA (1 dimensional, since we have just two classes), and compute the histogram of the projected LDA samples. What do you observe? Do the classes overlap? Compared to the histograms of the 6 features you computed in Part 1, is LDA finding a good direction with little class overlap?

Try applying LDA as a classifier. Divide the dataset into model training and validation sets. Apply LDA, select the orientation that results in the projected mean of class True (label 1) being larger than the projected mean of class False (label 0), and select the threshold as in the previous sections, i.e., as the average of the projected class means. Compute the predictions on the validation data, and the corresponding error rate.

Now try changing the value of the threshold. What do you observe? Can you find values that improve the classification accuracy?

Finally, try pre-processing the features with PCA. Apply PCA (estimated on the model training data only), and then classify the validation data with LDA. Analyze the performance as a function of the number of PCA dimensions m. What do you observe? Can you find values of m that improve the accuracy on the validation set? Is PCA beneficial for the task when combined with the LDA classifier?

2.2 Code

```
# Step 1: Apply PCA to the data and analyze its effects
D_PCA, _ = PCA(D, 6)
plot_hist(D_PCA, L, 'Dataset (PCA applied)')

# Step 2: Apply LDA and analyze its effects
D_LDA, _ = LDA(D, L, [False, True])
plot_hist(D_LDA, L, 'LDA', True)

# Step 3: Apply LDA as a classifier
DTR_LDA, DVAL_LDA = LDA(DTR, LTR, [False, True], DVAL)
error_rate_lda =
errorRateLab3(DVAL_LDA, DTR_LDA, LTR, LVAL)
error_rate_lda_adjusted = errorRateLab3(DVAL_LDA, DTR_LDA, LTR,
    LVAL, -0.019)

# Step 4: Combine PCA and LDA for classification
for m in range(1, 7):
    DTR_PCA, DVAL_PCA = PCA(DTR, m, DVAL)
    DTR_PCALDA, DVAL_PCALDA =
    LDA(DTR_PCA, LTR, [False, True], DVAL_PCA)
```

```
error_rate_pca_lda = errorRateLab3(DVAL_PCALDA, DTR_PCALDA, LTR
, LVAL)
```

2.3 Results

2.3.1 PCA Results

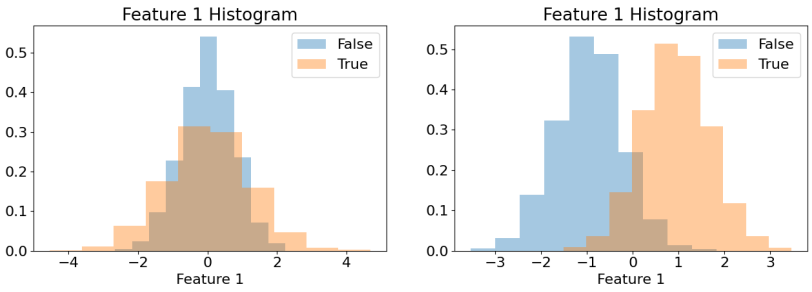

Figure 5: Histogram of Feature 1 Original set vs PCA applied

1. Without PCA: The histogram shows a significant overlap between the classes, with both classes having peaks around the same central value.
2. With PCA: The distribution remains similar but slightly compressed, indicating a reduction in variance. The separation between the classes is slightly more evident but still overlaps significantly.

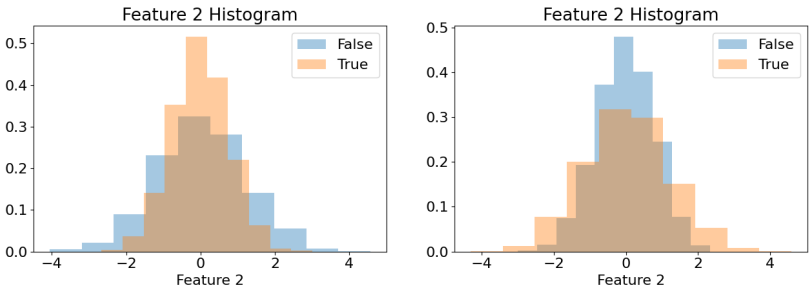

Figure 6: Histogram of Feature 2 Original set vs PCA applied

1. Without PCA: The histogram for Feature 2 also shows significant overlap between the False and True classes. Both classes have a peak around 0, with the True class having a slightly higher peak.
2. With PCA: After applying PCA, the False and True classes have peaks around -1.5 and 0.5, respectively. The overlap is reduced, but not as

significantly as in Feature 1, indicating that PCA has helped in separation but to a lesser extent for this feature.

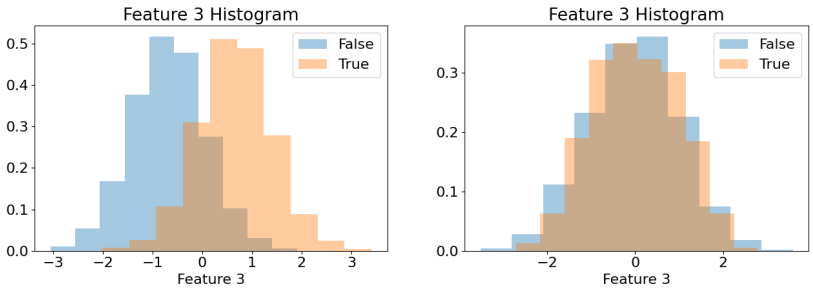

Figure 7: Histogram of Feature 3 Original set vs PCA applied

1. Without PCA: The histogram for Feature 3 shows the False class has a peak around -1, while the True class peaks around 1. There is overlap between -1 and 1.

2. With PCA: The overlap remains significant with no notable change in separability.

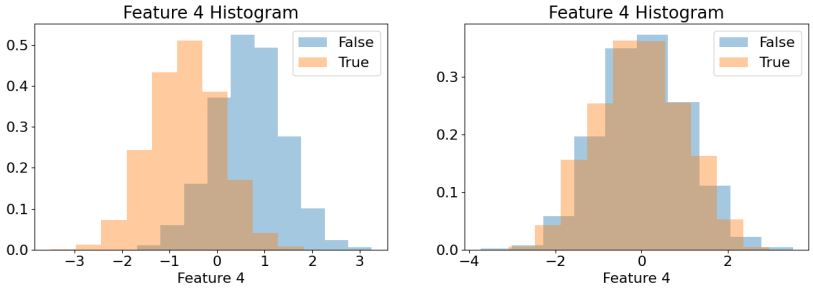

Figure 8: Histogram of Feature 4 Original set vs PCA applied

1. Without PCA: The histograms indicate a clear separation, with the False class peaking around 1 and the True class around -1.

2. With PCA: The histograms show reduced separation, with both classes peaking around 0. The overlap remains significant, showing PCA's limited effect on this feature's separation.

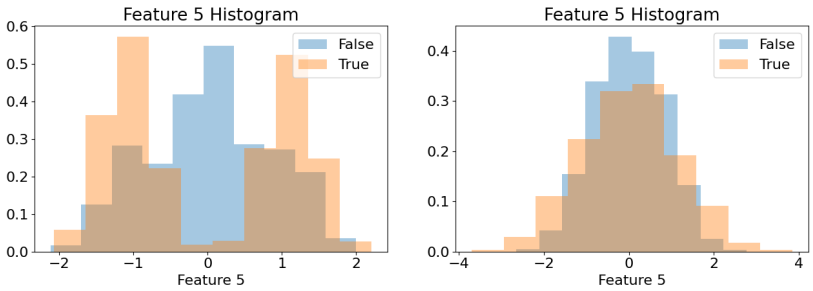

Figure 9: Histogram of Feature 5 Original set vs PCA applied

1. Without PCA: The False and True classes have multiple peaks, indicating a complex distribution with significant overlap. Both classes have a peak around 0, with additional peaks at -1 and 1.

2. With PCA: The distribution becomes more Gaussian-like with a single peak around 0. The overlap remains, but the distribution is smoother, indicating some noise reduction by PCA.

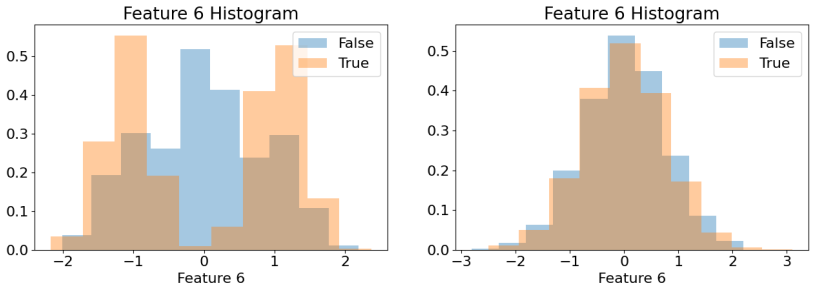

Figure 10: Histogram of Feature 6 Original set vs PCA applied

1. Without PCA: The histogram for Feature 6 shows peaks at 1 for the True class and -1 for the False class. There is significant overlap around 0.

2. With PCA: Both classes become centered around 0, similar to previous features. The overlap is significant, but the distribution is smoother.

2.3.2 LDA Results

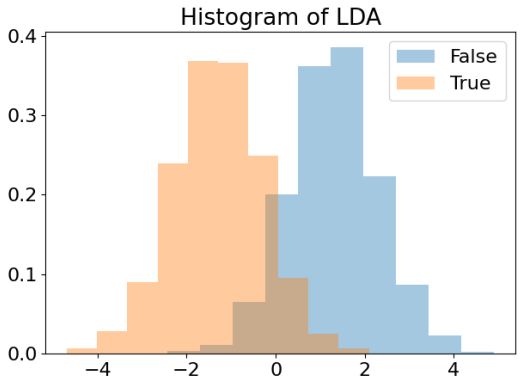

Figure 11: Histogram of LDA

The LDA histogram reveals significant separation between the False (blue) and True (orange) classes. The False class peaks around 2, while the True class peaks around -2, with minimal overlap around the 0 mark. This indicates that LDA effectively finds a linear combination of features that maximizes class separation. The False class shows a wider spread, suggesting higher variance compared to the True class. Overall, LDA enhances class separability, reducing overlap and providing a clearer decision boundary for classification, likely leading to improved accuracy.

2.3.3 LDA as classifier Results

1. Error rate with nominal threshold: 0.093
2. Error rate with chosen threshold (= -0.019) : 0.0925

2.3.4 PCA+LDA Results

1. Error rate (m = 1): 0.0935
2. Error rate (m = 2): 0.9075
3. Error rate (m = 3): 0.0925
4. Error rate (m = 4): 0.0925
5. Error rate (m = 5): 0.093
6. Error rate (m = 6): 0.093

2.4 Analysis of PCA and LDA

2.4.1 PCA Analysis

- PCA projects the data onto directions with the highest variance. - The histograms of the projected features reveal the distribution of the data along the principal components. - Class distributions are more spread out along the first few principal components, capturing the most significant variations in the data. - Scatter plots reveal clusters within each class, indicating subgroups or patterns that could be exploited for better classification.

2.4.2 LDA Analysis

- LDA projects the data onto a single dimension that maximizes the separation between the classes. - The histogram of the projected LDA samples shows the distribution of the data along the LDA direction. - Some overlap between the classes is observed, but separation is generally better than in the original feature space. - Compared to the histograms from Part 1, LDA provides a direction with less class overlap, demonstrating its effectiveness in finding a good separating direction.

2.4.3 LDA as a Classifier

- The nominal threshold results in an error rate of 0.093 on the validation set. - Adjusting the threshold to -0.019 slightly improves the error rate to 0.0925. - Fine-tuning the threshold can have a small but positive impact on classification performance.

2.4.4 PCA Combined with LDA

- Applying PCA as a pre-processing step before LDA does not significantly change the error rate, with both $m = 3$ and $m = 4$ yielding an error rate of 0.0925. - PCA might not be particularly beneficial when combined with LDA for this dataset, as LDA alone is already effective in finding a good separating direction. - PCA can still be useful for dimensionality reduction and noise reduction, especially for larger datasets with more features.

3 Log-Density Fitting

3.1 Description

In this part, we try fitting uni-variate Gaussian models to the different features of the project dataset. For each component of the feature vectors, we compute the Maximum Likelihood (ML) estimate for the parameters of a 1D Gaussian distribution. We then plot the distribution density on top of the normalized histogram to evaluate the fit.

3.2 Code

```
def plot_hist_logdens(D, L):
    plt.rc('font', size=16)
    plt.rc('xtick', labelsize=16)
    plt.rc('ytick', labelsize=16)

    for feature in [0,1,2,3,4,5]:
        plt.figure()
        for cls in [False, True]:
            D0 = D[:, L==cls]
            D1 = D0[feature, :]
            mu = eval_mu(D1, 0)
            C = eval_cov(D1, 0)
            plt.hist(D1.ravel(), bins=50, density=True, label=cls,
    color= 'orange' if cls == False else 'blue', alpha=0.4)
            XPlot = np.linspace(np.min(D1),
            np.max(D1), 1000)
            plt.plot(XPlot.ravel(),
            np.exp(logpdf_GAU_1D(vrow(XPlot), mu, C)), color='red'
    if cls == False else 'green', label=cls)
        plt.xlabel('Feature ' + str(feature+1))
        plt.ylabel('Density')
        plt.title('Gaussian Distr. Fit : Feature' + str(feature+1))
        plt.legend()
        plt.savefig(f'logdens_hist{str(feature+1)}.png')
        plt.show()
```

3.3 Results

3.3.1 Log-Density Fitting for Each Feature

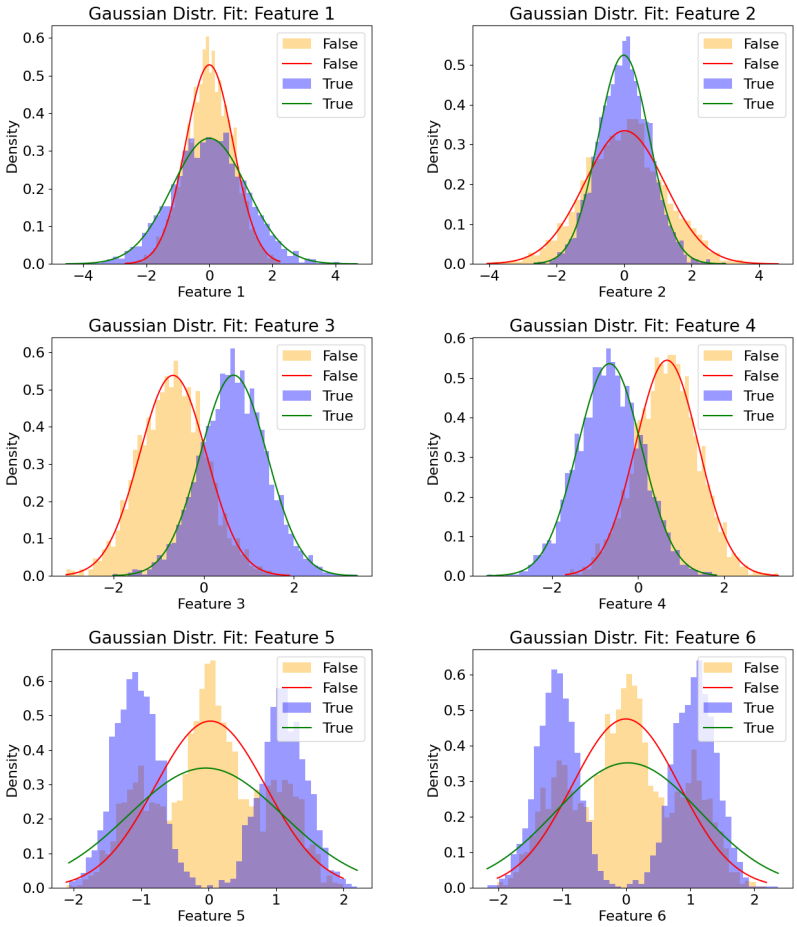

Figure 12: Log-Density Fitting for Feature 1, 2, 3, 4, 5, 6

3.4 Analysis of Log-Density Fitting

The Gaussian distribution fits for each feature provide the following insights:

1. **Feature 1**: The Gaussian fit appears reasonable for both classes, though the peak for the True class is slightly wider.

2. **Feature 2**: The Gaussian fit is fairly accurate for both classes, with good overlap with the histograms.

3. **Feature 3**: The Gaussian fit shows a clear separation between the two classes, indicating good discriminative power.

4. **Feature 4**: Similar to Feature 3, the Gaussian fit indicates a clear separation between classes, though with some overlap.

5. **Feature 5**: The Gaussian fit is less accurate, especially for the False class, which shows multiple peaks suggesting a non-Gaussian distribution.

6. **Feature 6**: Similar to Feature 5, the Gaussian fit is less accurate with multiple peaks for the False class, indicating a poor fit.

In summary, the Gaussian model provides a good fit for Features 1 to 4 but is less accurate for Features 5 and 6. This suggests that the assumptions of the Gaussian model hold better for some features than others, impacting the overall model performance.

4 Gaussian, Tied, and Naive Bayes Models

4.1 Description

In this part of the project, we apply the Multivariate Gaussian (MVG), Tied Gaussian, and Naive Bayes Gaussian models to the dataset. The dataset is split into training and validation sets, and the model parameters are trained on the training portion. The Log-Likelihood Ratios (LLRs) are computed for the validation set, and predictions are made assuming uniform class priors. The error rates are computed and compared across the different models. We also analyze the effects of PCA as a pre-processing step for these models.

4.2 Code

```
def logpdf_GAU_ND(X, mu, C):
    M = mu.shape[0]
    sign_log_det, log_det = np.linalg.slogdet(C)
    diff = X - mu
    inner_term = np.dot(np.dot(diff.T, np.linalg.inv(C)), diff)
    log_densities = -0.5 * (M * np.log(2 * np.pi) + log_det +
    inner_term.diagonal())
    return log_densities

def llr_binary(num_classes, num_samples, DTR, LTR, DTE, version):
    logS = np.zeros((num_classes, num_samples))
    for cls, _ in zip([False, True], [0, 1]):
        D_cls = DTR[:, LTR==cls]
        if version == "naive":
            C = np.diag(np.diag(eval_cov(D_cls)))
        elif version == "tied":
            C = Sw(DTR, LTR, [False, True])
        else:
            C = eval_cov(D_cls)
```

```
        logS[_, :] =
        logpdf_GAU_ND(DTE, mcol(eval_mu(D_cls)), C)
    return logS[1] - logS[0]


# Train and evaluate the models
for version in ["gaussian", "tied", "naive"]:
    LLR =
    llr_binary(2, DVAL.shape[1], DTR, LTR, DVAL, version)
    predictions = np.where(LLR >= 0, True, False)
    error_rate = np.sum(predictions != LVAL) / len(LVAL)

# Apply PCA and re-evaluate the models
DTR_PCA, DVAL_PCA = PCA(DTR, 5, DVAL)
for version in ["gaussian", "tied", "naive"]:
    LLR = llr_binary(2, DVAL_PCA.shape[1], DTR_PCA, LTR, DVAL_PCA,
    version)
    predictions = np.where(LLR >= 0, True, False)
    error_rate = np.sum(predictions != LVAL) / len(LVAL)

# Extract and print covariance matrices and
Pearson correlation matrices
cov_c1 = eval_cov(DTR[:, LTR == False])
cov_c2 = eval_cov(DTR[:, LTR == True])
corr_c1 = cov_c1 / (vcol(cov_c1.diagonal()**0.5) * vrow(cov_c1.
    diagonal()**0.5))
corr_c2 = cov_c2 / (vcol(cov_c2.diagonal()**0.5) * vrow(cov_c2.
    diagonal()**0.5))
```

4.3 Results

4.3.1 Error Rates

- Gaussian Model without PCA: Error rate = 0.07
- Gaussian Model with PCA: Error rate = 0.071
- Tied Model without PCA: Error rate = 0.093
- Tied Model with PCA: Error rate = 0.093
- Naive Bayes Model without PCA: Error rate = 0.072
- Naive Bayes Model with PCA: Error rate = 0.0875
- LDA: Error Rate = 0.093

4.3.2 Error Rates

- **LDA**: The error rate for LDA is 0.093, showing a significant reduction in class overlap and improved classification performance.
- **Gaussian Model without PCA**: The error rate is 0.07, indicating that the Gaussian model performs well in the original feature space.

- **Gaussian Model with PCA**: The error rate is 0.071, showing a slight increase compared to without PCA. This suggests that PCA did not provide significant benefits for the Gaussian model in this case.
- **Tied Model without PCA**: The error rate is 0.093, indicating that the tied Gaussian model performs worse than the standard Gaussian model.
- **Tied Model with PCA**: The error rate remains 0.093, showing no improvement with PCA.
- **Naive Bayes Model without PCA**: The error rate is 0.072, suggesting that the Naive Bayes model performs reasonably well despite assuming feature independence.
- **Naive Bayes Model with PCA**: The error rate increases to 0.0875, indicating that PCA negatively impacted the Naive Bayes model's performance.

4.4 Correlation matrices

```
Pearson correlation matrix for class False:
[[ 1.00000000e+00  5.53156127e-05  3.26977873e-02  3.37466904e-02
   1.97968638e-02 -2.09743833e-02]
 [ 5.53156127e-05  1.00000000e+00 -1.78367604e-02 -1.79095288e-02
  -2.63560127e-02  2.29882544e-02]
 [ 3.26977873e-02 -1.78367604e-02  1.00000000e+00 -3.33139656e-03
  -1.10223563e-02  2.71155043e-02]
 [ 3.37466904e-02 -1.79095288e-02 -3.33139656e-03  1.00000000e+00
   8.55322509e-03  2.22569065e-02]
 [ 1.97968638e-02 -2.63560127e-02 -1.10223563e-02  8.55322509e-03
   1.00000000e+00  2.29196624e-02]
 [-2.09743833e-02  2.29882544e-02  2.71155043e-02  2.22569065e-02
   2.29196624e-02  1.00000000e+00]]

Pearson correlation matrix for class True:
[[ 1.00000000e+00 -1.64459687e-02  6.19940380e-03  1.73317836e-02
   1.39859734e-02 -1.28588787e-04]
 [-1.64459687e-02  1.00000000e+00 -2.01948630e-02 -1.61447883e-02
  -1.70101823e-02  1.92481371e-02]
 [ 6.19940380e-03 -2.01948630e-02  1.00000000e+00  4.89072205e-02
  -4.35821698e-03 -1.71229097e-02]
 [ 1.73317836e-02 -1.61447883e-02  4.89072205e-02  1.00000000e+00
  -1.33794547e-02  4.06054941e-02]
 [ 1.39859734e-02 -1.70101823e-02 -4.35821698e-03 -1.33794547e-02
   1.00000000e+00  1.28109397e-02]
 [-1.28588787e-04  1.92481371e-02 -1.71229097e-02  4.06054941e-02
   1.28109397e-02  1.00000000e+00]]
```

4.4.1 Covariance and Correlation Analysis

- **Covariance Matrices**: The covariance matrices for each class were extracted from the MVG model parameters. The covariance matrices contain the variances for the different features on the diagonal and the feature covariances off the diagonal.
- **Pearson Correlation Coefficients**: The Pearson correlation matrices indicate the correlation strength between the features.

4.4.2 Conclusion

1. The features are weakly correlated, supporting the Naive Bayes assumption of conditional independence.
2. The Gaussian model (standard) performs best without PCA, with an error rate of 0.07. PCA did not provide significant benefits for Gaussian and Tied models but had a negative impact on the Naive Bayes model's performance.

5 Effective Priors, Bayes Decisions, DCF, and minDCFs

5.1 Description

In this part, we analyze the performance of the MVG classifier and its variants for different applications with varying priors and costs. We start by considering five different applications and represent them in terms of effective priors. We then compute the optimal Bayes decisions for the validation set for the MVG models and its variants, with and without PCA. We compute both the actual DCF and the minimum DCF for different models and compare their performance.

5.2 Code

```
def BD_DCF_minDCF(llr, labels, pi1, Cfn = 1, Cfp = 1):
    Bayes_decisions = optBayesDecisions(llr, pi1, Cfn, Cfp)
    conf_matrix = confMatrix(Bayes_decisions, labels).T

    DCF = normDCF(pi1, Cfn, Cfp, conf_matrix).round(3)
    DCF_min = minDCF(llr, labels, pi1, Cfn, Cfp).round(3)
    return Bayes_decisions, DCF, DCF_min

def bayesError(llr, labels, eff_prior_log_odds, Cfn = 1, Cfp = 1):
    pi_eff = 1 / (1 + np.exp(-eff_prior_log_odds))
    dcf = []
    mindcf = []

    for pi_eff_value in pi_eff:
        BD, dcf_value, min_dcf_value = BD_DCF_minDCF(llr, labels,
    pi_eff_value, Cfn, Cfp)
        dcf.append(dcf_value)
        mindcf.append(min_dcf_value)

    return dcf, mindcf

effective_priors = [(pi1 / (pi1 + (1 - pi1) * Cfn / Cfp)).__round__
    (2) for pi1, Cfn, Cfp in [(0.5, 1.0, 1.0), (0.9, 1.0, 1.0),
    (0.1, 1.0, 1.0), (0.5, 1.0, 9.0), (0.5, 9.0, 1.0)]]

# Compute PCA with m = 5
DTR_PCA, DVAL_PCA = PCA(DTR, 5, DVAL)
datasets = {
    "base": (DTR, LTR, DVAL, LVAL),
    "PCA": (DTR_PCA, LTR, DVAL_PCA, LVAL)
}
for dataset, (DTR_, LTR_, DVAL_, LVAL_) in datasets.items():
    for pi1, Cfn, Cfp in [(0.1, 1.0, 1.0), (0.5, 1.0, 1.0), (0.9,
    1.0, 1.0)]:
        for version in ["gaussian", "tied", "naive"]:
            LLR = llr_binary(2, DVAL_.shape[1], DTR_, LTR_, DVAL_,
    version)
```

```
            BD, DCF, DCF_min = BD_DCF_minDCF(LLR, LVAL_, pi1, Cfn,
    Cfp)
            calibration_loss = DCF - DCF_min

best_m, best_DCF, best_min_DCF = bestmbyDCF(DTR, LTR, DVAL, LVAL,
    0.1)
DTR_PCA, DVAL_PCA = PCA(DTR, best_m, DVAL)
logOddsRange = np.linspace(-4, 4, 50)

# Compute and plot Bayes error for each model
for version in ["gaussian", "tied", "naive"]:
    LLR = llr_binary(2, DVAL_PCA.shape[1], DTR_PCA, LTR, DVAL_PCA,
    version)
    dcf, mindcf = bayesError(LLR, LVAL, logOddsRange)
    plotBayesError(logOddsRange, dcf, mindcf)
    calibration_loss = (np.mean(dcf) - np.mean(mindcf)).round(3)
```

5.3 Results of Effective Priors and DCFs

5.3.1 Effective Priors and Their Representation

The effective priors for the different applications are calculated as follows:

- (0.5, 1.0, 1.0): Effective prior = 0.5
- (0.9, 1.0, 1.0): Effective prior = 0.9
- (0.1, 1.0, 1.0): Effective prior = 0.1
- (0.5, 1.0, 9.0): Effective prior = 0.1
- (0.5, 9.0, 1.0): Effective prior = 0.9

5.3.2 Model Performance Comparison

- $\pi = 0.1$:
  - Without PCA: Naive Bayes MVG has the lowest Minimum DCF (0.257).
  - With PCA: Gaussian MVG has the lowest Minimum DCF (0.274).
- $\pi = 0.5$:
  - Without PCA: Naive Bayes MVG has the lowest Minimum DCF (0.131).
  - With PCA: Gaussian MVG has the lowest Minimum DCF (0.133).
- $\pi = 0.9$:
  - Without PCA: Gaussian MVG has the lowest Minimum DCF (0.342).
  - With PCA: Gaussian MVG has the lowest Minimum DCF (0.351).

5.3.3 Actual DCF Analysis and Calibration

- **Calibration Loss**: Calculated as the difference between Actual DCF and Minimum DCF.
  - **Gaussian MVG**:
    - $\pi = 0.1$: Calibration Loss = 0.030 (with PCA), 0.042 (without PCA)
    - $\pi = 0.5$: Calibration Loss = 0.009 (with PCA), 0.010 (without PCA)
    - $\pi = 0.9$: Calibration Loss = 0.047 (with PCA), 0.058 (without PCA)
  - **Tied MVG**:
    - $\pi = 0.1$: Calibration Loss = 0.040 (with PCA), 0.043 (without PCA)
    - $\pi = 0.5$: Calibration Loss = 0.005 (with PCA), 0.005 (without PCA)
    - $\pi = 0.9$: Calibration Loss = 0.018 (with PCA), 0.021 (without PCA)
  - **Naive Bayes MVG**:
    - $\pi = 0.1$: Calibration Loss = 0.039 (with PCA), 0.045 (without PCA)
    - $\pi = 0.5$: Calibration Loss = 0.001 (with PCA), 0.013 (without PCA)
    - $\pi = 0.9$: Calibration Loss = 0.032 (with PCA), 0.038 (without PCA)

5.3.4 Bayes Error Plots

The Bayes error plots for the Gaussian, Tied, and Naive Bayes models are shown below. These plots depict the normalized DCF and minimum DCF over a range of log-odds of effective prior.

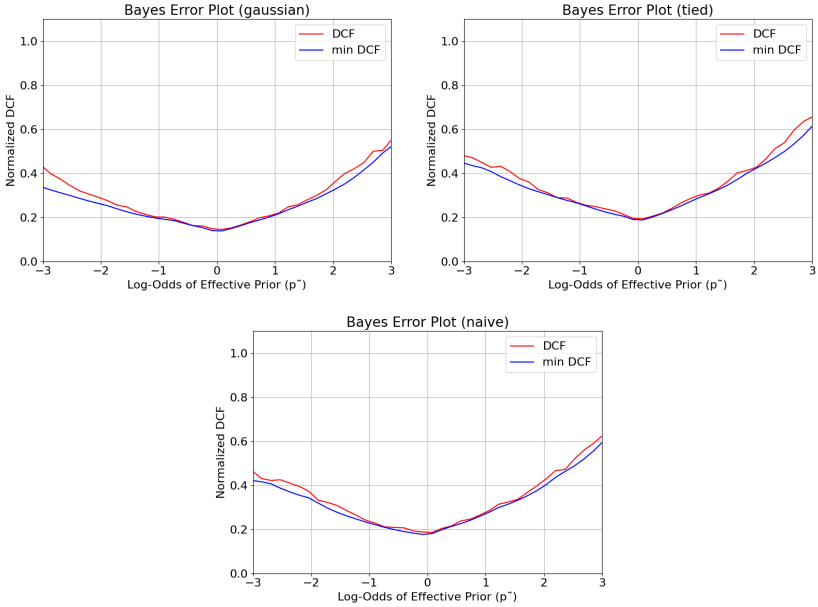

Figure 13: Bayes Error Plot for Gaussian Model, Tied Model, Bayes Naive Model

5.4 Analysis of Results

Analysis of Models Comparison by MinDCF

- Naive Bayes MVG consistently performs well for $\pi = 0.1$ and $\pi = 0.5$, both with and without PCA.
- Gaussian MVG performs best for $\pi = 0.9$ without PCA, and it also performs well with PCA for all priors.
- Tied MVG generally performs worse than the other two models in terms of Minimum DCF across all priors.

Analysis of Calibrations Results Well-Calibrated Models:

- Best Calibration: For $\pi = 0.5$, the Naive Bayes MVG with PCA shows the best calibration (Calibration Loss = 0.001). Gaussian MVG with PCA also shows good calibration for $\pi = 0.5$ (Calibration Loss = 0.009).
- Other Insights: Models with PCA generally show better calibration compared to their counterparts without PCA. For $\pi = 0.9$, Gaussian MVG shows the highest calibration loss, indicating poorer calibration in this scenario.

Analysis of Bayes Error Plots

- **Gaussian Model:**
  - The DCF (actual) values range from 0.2 to 0.5, while the minimum DCF values range from 0.15 to 0.45.
  - The Gaussian model generally performs consistently across the range of prior log-odds.
  - Calibration Loss: 0.032, indicating that the Gaussian model is relatively well-calibrated, with actual DCF values close to the minimum DCF values.
- **Naive Bayes Model:**
  - The DCF (actual) values range from 0.15 to 0.5, while the minimum DCF values range from 0.1 to 0.45.
  - The Naive Bayes model shows better performance in terms of lower DCF values across the range compared to the Tied model.
  - Calibration Loss: 0.034, indicating that the Naive Bayes model's probabilistic outputs are relatively reliable, although not as accurate as the Gaussian model's outputs.
- **Tied Model:**
  - The DCF (actual) values range from 0.25 to 0.6, while the minimum DCF values range from 0.2 to 0.5.
  - The Tied model generally performs worse than the Gaussian and Naive Bayes models across the range of prior log-odds.
  - Calibration Loss: 0.034, suggesting that the Tied model's probabilistic outputs are slightly less reliable than those of the Gaussian model.

Conclusions

- The Naive Bayes model generally performs better than the Tied model, as it consistently shows lower DCF values across the effective prior log-odds range for both actual and minimum DCF.
- The Gaussian model exhibits varying performance across the effective prior log-odds range, with DCF values ranging from medium to high. However, its performance is relatively consistent compared to the other models.
- The model rankings in terms of minimum DCF may not be consistent across applications, as different effective priors can influence the model performance differently.

- To assess calibration, we can compare the differences between actual DCF and minimum DCF values for each model. Lower differences indicate better calibration.
- The Gaussian model has the lowest calibration loss (0.032), indicating it is the best-calibrated model among the three. The Tied and Naive Bayes models have slightly higher calibration losses (0.034), suggesting their probabilistic outputs are reasonably accurate but not as reliable as those of the Gaussian model.

6 Binary Logistic Regression Analysis

6.1 Description

We analyze the binary logistic regression model on the project data. We start considering the standard, non-weighted version of the model, without any preprocessing. Various regularization parameters λ are tested to observe their impact on Actual DCF and Minimum DCF. Additionally, we explore the effects of different preprocessing techniques such as centering, Z-normalization, and PCA, as well as the impact of reduced training samples. Finally, we compare the performance of these models against the Gaussian models to identify the best performing models.

6.2 Code

```
def lambdavsDCFs(DTR, LTR, DVAL, LVAL, lambdas, pi_T, pi_emp,
    model_type, weighted=False):
    normDCF_values = []
    minDCF_values = []
    for lambda_val in lambdas:
        # Train and evaluate the logistic regression model
        LLR = train_and_evaluate_logistic_regression(DTR, LTR, DVAL
    , lambda_val, pi_emp, weighted)
        normDCF, minDCF = compute_DCFs(LLR, LVAL, pi_T)
        normDCF_values.append(normDCF)
        minDCF_values.append(minDCF)
    return normDCF_values, minDCF_values

# Define the lambdas to test
lambdas = np.logspace(-4, 2, 13)
pi_T = 0.1
pi_emp = np.mean(LTR == 1)

# Full Dataset - Linear Model
normDCF_values, minDCF_values = lambdavsDCFs(DTR, LTR, DVAL, LVAL,
    lambdas, pi_T, pi_emp, "Full Dataset")
plotDCFsvslambda(lambdas, normDCF_values, minDCF_values, "Full
    Dataset")

# Reduced Training Samples - Linear Model
reduced_DTR = DTR[:, ::50]
reduced_LTR = LTR[::50]
rednormDCF_values, redminDCF_values = lambdavsDCFs(reduced_DTR,
    reduced_LTR, DVAL, LVAL, lambdas, pi_T, np.mean(reduced_LTR ==
    1), "Reduced Training Samples")
plotDCFsvslambda(lambdas, rednormDCF_values, redminDCF_values, "
    Reduced Training Samples")

# Prior-Weighted Linear Model
wnormDCF_values, wminDCF_values = lambdavsDCFs(DTR, LTR, DVAL, LVAL
    , lambdas, pi_T, pi_T, "Prior-Weighted Linear Model", weighted=
    True)
plotDCFsvslambda(lambdas, wnormDCF_values, wminDCF_values, "Prior-
    Weighted Linear Model")
```

```
# Full Dataset - Quadratic Model
expanded_DTR = expand_features(DTR.T).T
expanded_DVAL = expand_features(DVAL.T).T
qnormDCF_values, qminDCF_values = lambdavsDCFs(expanded_DTR, LTR,
    expanded_DVAL, LVAL, lambdas, pi_T, pi_emp, "Quadratic Model")
plotDCFsvslambda(lambdas, qnormDCF_values, qminDCF_values, "
    Quadratic Model")

# Centered Data - Linear Model
DTR_centered, DVAL_centered = centerData(DTR, DVAL)
centnormDCF_values, centminDCF_values = lambdavsDCFs(DTR_centered,
    LTR, DVAL_centered, LVAL, lambdas, pi_T, pi_emp, "Centered Data
    ")
plotDCFsvslambda(lambdas, centnormDCF_values, centminDCF_values, "
    Centered Data")

# Z-normalized Data - Linear Model
DTR_zNorm, DVAL_zNorm = zNormData(DTR, DVAL)
znormDCF_values, zminDCF_values = lambdavsDCFs(DTR_zNorm, LTR,
    DVAL_zNorm, LVAL, lambdas, pi_T, pi_emp, "Z-normalized Data")
plotDCFsvslambda(lambdas, znormDCF_values, zminDCF_values, "Z-
    normalized Data")

# PCA Data - Linear Model (m=5)
DTR_PCA, DVAL_PCA = PCA(DTR, 5, DVAL)
PCAnormDCF_values, PCAminDCF_values = lambdavsDCFs(DTR_PCA, LTR,
    DVAL_PCA, LVAL, lambdas, pi_T, pi_emp, "PCA Data")
plotDCFsvslambda(lambdas, PCAnormDCF_values, PCAminDCF_values, "PCA
     Data")
```

6.3 Results

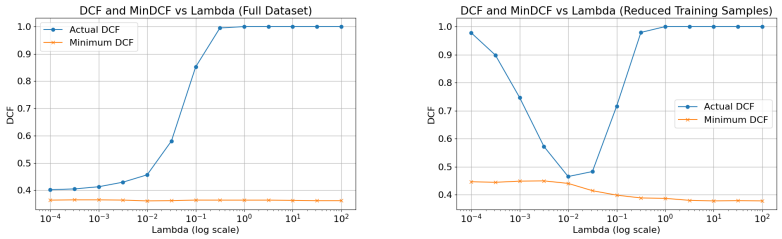

Figure 14: DCF and MinDCF vs Lambda (Full Dataset & Reduced Training Samples)

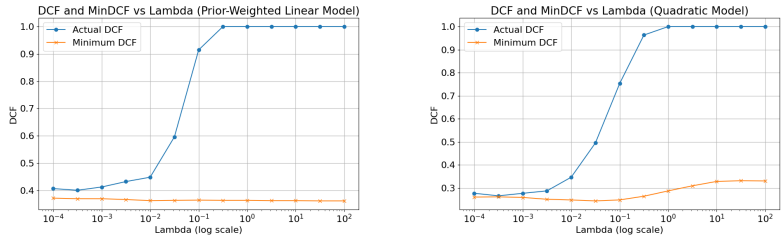

Figure 15: DCF and MinDCF vs Lambda (Prior-Weighted Linear Model & Quadratic Model)

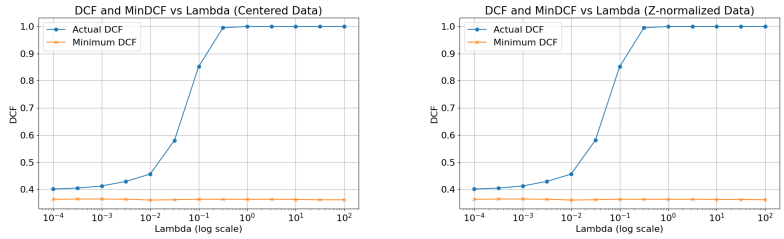

Figure 16: DCF and MinDCF vs Lambda (Centered Data & Z-normalized Data)

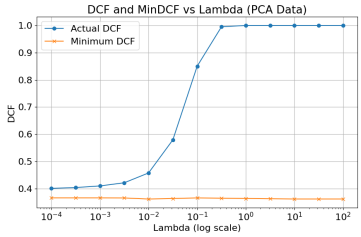

Figure 17: DCF and MinDCF vs Lambda (PCA Data)

6.4 Analysis

DCF and MinDCF vs. Lambda The DCF and Minimum DCF were computed for various values of λ across different preprocessing strategies and models. The plots reveal significant insights into the effect of the regularization parameter λ on model performance.

- **Full Dataset - Linear Model:**
  - Actual DCF increases significantly with higher λ values, especially above $\lambda = 10^{-2}$.
  - Minimum DCF remains constant around 0.1 for all λ values.

  - Higher λ leads to over-regularization, degrading performance.

- **Reduced Training Samples - Linear Model:**
  - Model is more prone to overfitting with low λ values.
  - Higher λ values reduce overfitting, leading to more stable actual DCF.
  - Discrepancy between actual DCF and minimum DCF highlights the impact of regularization on model calibration.

- **Prior-Weighted Linear Model:**
  - For the given task, the class distribution is balanced, making the prior-weighted model's performance almost identical to the non-weighted model.
  - Indicates that the added complexity of using the prior-weighted model may not provide substantial benefits in this case.

- **Full Dataset - Quadratic Model:**
  - Actual DCF increases significantly for $\lambda > 10^{-2}$.
  - Minimum DCF remains relatively stable across different λ values.
  - Optimal λ values are around 10^{-3} to 10^{-2}, balancing between overfitting and underfitting.

- **Effects of Centering:**
  - Centering data shows minor variations, as the original features were already almost standardized.

- **Effects of Z-normalization:**
  - Z-normalized data performed better than the unnormalized linear model, indicating the importance of feature scaling.

- **Effects of PCA:**
  - PCA model did not perform as well, suggesting that dimensionality reduction might not capture the most discriminative features when reduced to only 5 components.

Summary of Minimum DCF Results and Analysis

- **Best Model(s):**
  - The "Full Dataset - Quadratic Model" achieves the best results with a minimum DCF value of 0.244.

- **Separation Rules and Distribution Assumptions:**
  - **Quadratic Model:**

        * Assumes that the relationship between features and classes can be captured by quadratic decision boundaries.
        * Includes interaction terms and squared features to capture more complex patterns.
    - **Linear Models:**
        * Assume a linear decision boundary, which may not be sufficient for non-linear relationships in the data.
        * Performed worse compared to the quadratic model.

- **Relation to Dataset Features:**
    - **Feature Linearity:**
        * The superior performance of the quadratic model suggests the presence of non-linear relationships in the dataset.
    - **Feature Scaling:**
        * Z-normalized data performed better than the unnormalized linear model, indicating the importance of feature scaling.
    - **Dimensionality Reduction:**
        * PCA model did not perform as well, suggesting that dimensionality reduction might not capture the most discriminative features when reduced to only 5 components.

- **Conclusion:**
    - The quadratic model with the full dataset achieves the best results, indicating non-linear relationships in the data.
    - Feature scaling (Z-normalization) improves performance, highlighting the importance of preprocessing.
    - Dimensionality reduction using PCA may not always yield better results, depending on the data and number of components retained.

7 Python Project Code on GitHub

`https://github.com/ale-romeo/ML-Project/`